

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

DYNAMIC PROTOCOL CONSTRUCTION

Inventor(s):

Alfred Lee

David Levin

Erik B. Christensen

David Wortendyke

Saurab Nog

Donald F. Box

Christopher G. Kaler

Giovanni M. Della-Libera

ATTORNEY'S DOCKET NO. MS1-1861US

CLIENT'S DOCKET NO. 306888.01

DYNAMIC PROTOCOL CONSTRUCTION

RELATED APPLICATIONS

[0001] This patent application is related to co-owned U.S. Patent Application Serial No. _____ (Client/Attorney Docket Numbers MS306881.01/MS1-1853US), entitled "Invalid Policy Detection," and U.S. Patent Application Serial No. _____ (MS306882.01/MS1-1855US), entitled "Policy Application Across Multiple Nodes," both of which are hereby incorporated by reference for all that they disclose.

TECHNICAL FIELD

[0002] The described subject matter relates to electronic computing, and more particularly to systems and methods for dynamic protocol construction.

BACKGROUND

[0003] Communication between various computing devices (e.g., personal computers, server computers, mobile devices) is increasingly commonplace in a number of network environments, such as, e.g., the Internet and corporate intranets to name only a few examples. Often, these computing devices are configured for communication in accordance with preferred or even required protocols. Traditionally when a computing device attempts to engage in communication with

1 another computing device using an unrecognized protocol, an error message is sent
2 to the first device, and further communication typically cannot proceed.

3 [0004] As an illustration, a commercial web site may require a user's
4 computer to comply with a particular protocol or data format before the user is
5 granted access to the payment web pages. For example, the commercial website
6 may require that incoming messages be encoded according to a particular
7 encryption scheme for security purposes, or that incoming messages be formatted
8 using a particular compression scheme to facilitate efficient transaction processing.
9 If the user's computer is not equipped to abide by the specified protocol or data
10 format, the user's computer generally receives an error notification, such as a
11 "400" error code defined in the Hypertext Transport Protocol (HTTP). Typically,
12 such error notifications are not very informative or helpful for a user to remedy the
13 error, if possible, and continue communicating with the commercial website.
14

15 [0005] In addition, over time, as new protocols and data formatting
16 techniques emerge, not all computing devices will necessarily have adopted the
17 latest protocols and data formatting techniques. Thus, there will typically always
18 be some differences between the protocols and/or data formats used by some
19 computing devices and the protocols and/or data formats used by other computing
20 devices. However, although some computing devices may not be able to apply the
21 newest protocols or data formats, they typically can communicate using some other
22 protocols or data formats. Unfortunately, a traditional computing device does not
23
24
25

1 typically have the ability to identify the different protocols and/or data formats
2 used by another computing device, and adapt, if possible, to the different protocols
3 and/or data formats.
4

5 SUMMARY

6 [0006] Implementations are described and claimed herein to dynamically
7 construct a protocol to facilitate communication between nodes. Implementations
8 utilize policies associated with nodes to specify protocol properties of the nodes. A
9 policy expression in a policy related to a node can be selected by another node to
10 construct a protocol between the two nodes.
11

12 [0007] In some implementations, articles of manufacture are provided as
13 computer program products. One implementation of a computer program product
14 provides a computer program storage medium readable by a computer system and
15 encoding a computer program for dynamic protocol construction. Another
16 implementation of a computer program product may be provided in a computer
17 data signal embodied in a carrier wave by a computing system and encoding the
18 computer program for dynamic protocol construction.
19

20 [0008] The computer program product encodes a computer program for
21 executing on a computer system a computer process that generates a message
22 conforming to a group of assertions, the group of assertions characterizing
23 capabilities or requirements of a first node. The process may further include
24
25

1 sending a request to the first node for a policy including the group of assertions.

2 Alternatively, the process may further include retrieving the group of assertions
3 from a second node. The process may further include determining whether the
4 group of assertions related to the first node is compatible with a group of
5 assertions related to a second node.

6 [0009] In another implementation, a method includes advertising a policy
7 having assertions characterizing communication properties of a destination node,
8 each assertion specifying a communication property supported by the destination
9 node. Advertising can include generating a message including the policy in
10 response to a request for the policy. Advertising may also include incrementally
11 distributing the policy.
12

13 [0010] In yet another implementation, a system is provided including a
14 policy generator for generating at least one policy having assertions characterizing
15 properties of a node. The system may include a policy retriever retrieving a policy
16 from another node and a message generator generating a message to the other
17 node, wherein the message conforms to a group of assertions in the policy from the
18 other node.
19
20

21 **BRIEF DESCRIPTION OF THE DRAWINGS**

22
23 [0011] Fig. 1 illustrates an exemplary operating environment in which
24 dynamic protocol construction can be carried out;
25

1 [0012] Fig. 2 illustrates an exemplary policy including assertions that may
2 be used to construct a protocol for communication between two nodes;

3 [0013] Figs. 3-4 are flowcharts illustrating exemplary operations to
4 implement dynamic protocol construction; and

5 [0014] Fig. 5 is a schematic illustration of an exemplary computing device
6 that can be utilized to implement dynamic protocol construction.

7 8 DETAILED DESCRIPTION

9 Overview

10
11 [0015] Briefly, dynamic protocol construction may be implemented to
12 facilitate communication between two nodes. Because data communication
13 protocols and formats can change, communication between two nodes can be
14 seriously hampered by mismatches in the protocols and formats employed by each
15 of the nodes. The dynamic protocol construction scheme described herein allows a
16 node to generate a policy having statements (referred to as assertions) that
17 characterize properties of the node. The properties can relate to, for example,
18 capabilities and/or requirements of the node. Another node that attempts to
19 communicate with the first node can retrieve the policy and generate messages that
20 conform to the assertions given therein and thereby successfully communicate with
21 the node.
22
23
24
25

Exemplary System

1 [0016] Fig. 1 illustrates an exemplary operating environment 100 in which
2 dynamic protocol construction can be carried out. Two nodes, node A 102 and
3 node B 104, communicate with each other via a network 106. Node A 102 and
4 node B 104 may be arranged in any number of configurations. Typical
5 configurations are a client/server configuration or a peer-to-peer configuration.
6 The network 106 may include other intermediate nodes (not shown), through
7 which data pass during communication between node A 102 and node B. As such,
8 exemplary communication configurations can include 1 to N (i.e., single node to
9 multiple node) and N to N (i.e., multiple node to multiple node) arrangements.
10

12 [0017] In general, a node is a processing location in a computer network.
13 More particularly, in accordance with the various implementations described
14 herein, a node is a process or device that is uniquely addressable via a network. By
15 way of example, and not limitation, individually addressable computing devices,
16 groups or clusters of computing devices that have a common addressable
17 controller, addressable peripherals, such as addressable printers, and addressable
18 switches and routers, as well as processes executing on such devices, are all
19 examples of nodes.
20

21 [0018] The operating environment 100 supports many communication
22 scenarios that are frequently carried out over a network. Exemplary scenarios
23 include, but are not limited to, node A 102 accessing a resource from node B 104,
24
25

1 or node A 102 providing a service to node B 104. For example, a user of node B
2 104 may access a commercial Web site at node A 102 to buy books from the Web
3 site.

4 [0019] In the exemplary operating environment 100, data communication
5 between node A 102 and node B 104 is carried out by exchanging messages
6 between node A 102 and node B 104. When in a message exchange, node A 102
7 and node B 104 are designed to receive and/or transmit messages according to
8 certain data formats and/or follow certain protocols. Node A 102 and node B 104
9 each have policies that may be used to express the data formats and protocols that
10 can or should be used during message exchange.

12 [0020] More generally, a policy is an informal abstraction expressing
13 properties of a node. In the implementation of Fig. 1, a policy expression includes
14 one or more policy assertions (also referred to as 'assertions'). An assertion
15 represents an individual preference, requirement, capability, or other property that
16 a node (e.g., Node A 102) may, or in some circumstances, must comply with in
17 order to communicate with another node (e.g., Node B 104).

19 [0021] For example, node A 102 includes an A input policy 108 and an A
20 output policy 110. The A input policy 108 expresses one or more assertions related
21 to messages that are received by, or input to, node A. The A output policy 110
22 expresses one or more assertions related to messages that are transmitted, or output
23
24
25

1 by, node A. Similarly, node B 104 includes B input policy 112 and B output policy
2 114.

3 [0022] As shown in Fig. 1, the policies are illustrated as being implemented
4 in one or more documents; however, policies need not be stored in documents, but
5 rather, can be implemented in other forms, such as, stored in memory, dynamically
6 created or retrieved from another node, or otherwise. A policy may be expressed in
7 a markup language, such as, but not limited to, Hypertext Markup Language
8 (HTML) and Extensible Markup Language (XML). In addition, an input policy
9 and an output policy may be combined into a single policy.
10

11 [0023] To further illustrate the concept of a policy, a policy can specify
12 message encoding formats, security algorithms, tokens, transport addresses,
13 transaction semantics, routing requirements, and other properties related to
14 message transmission or reception. Implementations of policies described herein
15 specify one or more assertions, which can aid two nodes in a message exchange in
16 determining if their requirements and capabilities are compatible. The assertions
17 may be grouped and related to each other in some way. A group of one or more
18 assertions may be referred to as a policy expression.
19

20 [0024] Accordingly, A input policy 108 includes a number of groups of
21 input assertions, including a first policy expression 116 and a second policy
22 expression 118. Similarly, A output policy 110 includes a number of groups of
23 output assertions, including a first policy expression 120 and a second policy
24
25

1 expression 122. Likewise, B input policy 112 includes a number of groups of
2 input assertions, including a first policy expression 124 and a second policy
3 expression 126; and B output policy 114 includes a number of groups of output
4 assertions, including a first policy expression 128 and a second policy expression
5 130.

6 [0025] Expression (1) shown below illustrates how the assertions in A input
7 policy 108 can be related in a Boolean manner:

8
$$(1) \text{ } AInputPolicy: (A1 \otimes A2 \otimes A3) \oplus (A4 \otimes A5 \otimes A6) \dots$$

9

10 [0026] Expression (1) indicates that in order to comply with the A input
11 policy 108, a node attempting to send a message to node A can satisfy either
12 assertion A1, assertion A2, and assertion A3 together, or assertion A4, assertion
13 A5, and assertion A6 together, but typically not both groups of assertions. The
14 manner in which a node, such as node B 104, may use the A input policy 108 to
15 communicate with node A 102 is discussed further below. Other, non-Boolean,
16 expressions can be used to express relationships among assertions.
17

18 [0027] The number of assertions shown in policy expressions 116, 118, 120,
19 122, 124, 126, 128, and 130 is purely exemplary for illustrative purposes only. The
20 numbers assigned to the assertions shown in Fig. 1 (e.g., A1, A2, ..., B13) are not
21 intended to imply that the various assertions shown are different or the same.
22 Indeed, frequently during operation, some assertions at node A 102 will match
23 some assertions of node B 104, and some assertions at node A 102 will be different
24
25

1 from some assertions at node B 104. A particular example of assertions is shown
2 in Fig. 2, and is discussed further below.

3 [0028] Node A 102 includes a policy generator 132, a policy retriever 134,
4 and a message generator 136. The policy generator 132 generates the A input
5 policy 108 and the A output policy 110. The policy generator 132 can send either
6 or both of the A input policy 108 and/or the A output policy 110 to node B 104 or
7 other intermediate nodes in the network 106. One particular implementation of the
8 policy generator 132 advertises the A input policy 108 and/or the A output policy
9 110, for example, by making A input policy 108 and/or the A output policy 110
10 publicly available either on node A 102 or some other node on the network 106.
11

12 [0029] The policy retriever 134 retrieves policies from other nodes, such as
13 node B 104 or intermediate nodes on the network 106. The policy retriever 134
14 can request a policy from another node, receive the policy, and may cache a
15 received policy in memory for later use. The policy retriever 134 can also retrieve
16 a policy that was previously stored in local memory on node A 102. The policy
17 retriever 134 performs functions related to determining whether a retrieved policy
18 is compatible with a local policy and/or selecting a compatible policy expression in
19 a retrieved policy.
20

21 [0030] The message generator 136 at node A 102 generates messages that
22 conform to one or more assertions in the B input policy 112 of node B 104. For
23 example, the message generator 136 may encrypt, format, or encode a message as
24
25

1 specified by input assertions in the B input policy 112. As another example, the
2 message generator 136 may transmit the message according to a compliant
3 protocol (e.g., SOAP 1.1) specified in the B input policy 112. As yet another
4 example, the message generator 136 may apply a user signature or password to the
5 message in accordance with the B input policy 112. The output of the message
6 generator 136 is a policy-compliant message complying with the B input policy
7 112.

8 [0031] Similarly, node B 104 includes a policy generator 138, a policy
9 retriever 140, and a message generator 142. The policy generator 138 has
10 functionality similar to that of the policy generator 132 in node A 102. Thus, if
11 node A's 102 policy retriever 134 requests a policy from node B 104, node B's 104
12 policy generator 138 can responsively transmit one or more of the B input policy
13 114 and the B output policy 116 to the policy retriever 134 at node A 102.

14 [0032] The policy retriever 140 at node B 104 has functionality similar to
15 the functionality described above with respect to policy retriever 134 at node A
16 102. The message generator 142 at node B 104 formats and transmits messages to
17 node A 102 in accordance with one or more assertions in the input policy 108 of
18 node A 102.

19 [0033] Node A 102 can retrieve and use a policy of node B 104 to construct
20 a protocol with which to communicate to node B 104, and vice versa. This may
21 involve a selection process where a node selects one group of assertions from the
22
23
24
25

1 policy of the other node. For example, node B 104 retrieves (via the retriever 138)
2 and analyzes the A input policy 108 to determine if node B can comply with at
3 least one of the policy expressions, the first policy expression 116, the second
4 policy expression 118, etc., in the A input policy 108. The determination may
5 involve solving a relational equation such as expression (1) above. Other
6 exemplary methods for determining whether node B 104 can comply is discussed
7 below with respect to operations shown in Fig. 3.

8 [0034] Another implementation of the operating environment 100 includes a
9 third party service or tool that compares policies of two or more nodes to
10 determine whether they are compatible. Such a service or tool may operate on
11 node A 102, node B 104, or an intermediate node on the network 106. Thus, a
12 service may read B output policy 114 and read A input policy 108 and determine if
13 the policies are compatible. For example, the service may determine that the
14 policy expression 116 is compatible with the policy expression 128. The service
15 can notify node A 102 and node B 104 as to the results of the compatibility
16 determination.
17
18

19 [0035] Fig. 2 illustrates an exemplary policy 200 that may be used by a node
20 to dynamically construct a protocol to facilitate communication with one or more
21 other nodes. The exemplary policy 200 is in Extensible Markup Language (XML).
22 As such, the exemplary policy 200 includes a number of tags, starting with an open
23 bracket (<) and ending with a close bracket (>).
24
25

1 [0036] As discussed above, a policy includes one or more assertions that can
2 be grouped into one or more policy expressions. Grouping assertions can involve
3 applying a relationship operator to the group. A relationship operator specifies a
4 relationship between or among assertions in a group. Various other attributes,
5 assertion types, and operators can be applied to an assertion. The exemplary policy
6 200 illustrates just a few exemplary attributes, assertion types, and operators.
7 Other exemplary attributes, assertion types, and operators are discussed further
8 below.

9 [0037] The exemplary policy 200 includes two policy expressions bounded
10 by a `<wsp:ExactlyOne>` operator 202. A first policy expression 204 expresses a
11 security profile (i.e., `<wsse:SecurityToken>`) consisting of security specific
12 policy assertions. As shown in Fig. 2, the first policy expression 204 specifies
13 “Kerberos Authentication” (i.e., `<wsse:TokenType>wsse:Kerberosv5TGT`
14 `</wsse:TokenType>`) and “Privacy” (i.e., `<wssx:Privacy />`).
15

16 [0038] A second policy expression 206 specifies password authentication
17 (`<wsse:TokenType>wsse:UsernameToken</wsse:TokenType>`), an
18 integrity algorithm (i.e., `<wsse:Algorithm Type="wsse:AlgEncryption"`
19 `URI="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>`), and an audit
20 trail (i.e., `<wssx:Audit/>`). The integrity algorithm specifies a particular
21 encryption algorithm along with a Uniform Resource Identifier (URI) indicating a
22 network location from which the encryption algorithm can be obtained.
23
24
25

1 [0039] The <wsp:ExactlyOne> operator 202 bounding the first policy
2 expression 204 and the second policy expression 206 indicates that one and only
3 one of the groups of assertions can be selected; i.e., the first policy expression 204
4 and the second policy expression 206 are alternatives.

5 [0040] Bounding the first policy expression 204 is an "All" operator 208.
6 The All operator 208 indicates that all of the assertions in policy expression 204
7 must be practiced by a node if the policy expression 204 is selected. Similarly, the
8 second policy expression 206 is bounded by another "All" operator 210, which
9 indicates that all of the assertions in the second group 206 must be practiced if the
10 second policy expression 206 is selected.

12 [0041] Each assertion may be associated with a usage type or attribute. The
13 usage attribute stipulates how the assertion should be interpreted in relation to the
14 overall policy. To illustrate, a privacy assertion could, for example, specify that
15 privacy guarantees will be provided for information exchanged between two Web
16 services, while an encryption assertion could specify a requirement for encryption.
17 The privacy assertion and the encryption assertion differ, in that the privacy
18 assertion has no externally visible manifestation, while the encryption assertion is
19 externally manifested (i.e., the encryption assertion indicates a requirement on
20 messages being sent to and from the Web services). The privacy assertion is
21 simply a declaration that the Web services will guarantee some level of privacy to
22 the sender, while the encryption assertion requires cooperation between the two
23
24
25

Web services. Because usage can differ between assertions, a usage attribute can be used to characterize the difference. Various exemplary usage attributes are discussed below.

[0042] Accordingly, within the All operator 208 tag, and the All operator 210 tag, usage attributes indicate that the bounded assertions are “required”. In an alternative implementation, each assertion tag bounded by the All operator 208, and the All operator 210, could individually specify the usage attribute.

[0043] Also in the All operator 208 tag, and the All operator 210 tag, preference values are shown that indicate a level of preference of the corresponding groups. In the exemplary policy 200, the preference value of the first policy expression 204 is “100”, while the preference value for the second policy expression 206 is “1”, meaning that the first policy expression 204 is preferred over the second policy expression 206.

[0044] To capture the nature of differences among various assertions, five exemplary usage attributes are used in one particular implementation of a policy: Required, Optional, Rejected, Observed and Ignored. These exemplary usage attributes are shown and described below in Table 1:

Table 1: Exemplary Usage Attributes

Attribute	Meaning
Required	The assertion must be applied to the subject. If the subject does not meet the criteria expressed in the assertion a fault or error will occur.
Rejected	The assertion is explicitly not supported and if present will cause failure.

Optional	The assertion may be made of the subject but it is not required to be applied.
Observed	The assertion will be applied to all subjects and requesters of the service are informed that the policy will be applied.
Ignored	The assertion is processed, but ignored. That is, it can be specified, but no action will be taken as a result of it being specified. Subjects and requesters are informed that the policy will be ignored.

[0045] With regard to Table 1, a policy subject is a node to which a policy can be bound. Other exemplary operators and containers, in addition to the “All” operator and the “ExactlyOne” operator, are shown and described below in Table 2:

Table 2: Exemplary Assertion Operators/Containers

Operator/Container	Meaning
Policy	A policy expression that is the top level container for the set of policy operators and assertions.
ExactlyOne	An ExactlyOne operator may contain one or more policy assertions, references, or operators. The ExactlyOne operator requires that exactly one of the bounded operands be satisfied.
All	The All operator may contain one or more policy assertions, references, or operators. The All operator requires that every one of the bounded operands be satisfied.
OneOrMore	The OneOrMore operator may contain one or more policy assertions, references, or operators. The OneOrMore operator requires that at least one of the bounded operands be satisfied.

[0046] The exemplary attributes, operators, and containers described in Table 1 and Table 2 are in no way intended to limit a particular policy implementation to the attributes, operators, and containers shown. Those skilled in

1 the art may readily recognize and develop other attributes, operators, and
2 containers that are useful in a particular implementation, which are within the
3 scope of the present application.

4 [0047] Some examples of assertions that may be made related to protocols
5 are Simple Object Access Protocol (SOAP) 1.1, SOAP 1.0, HyperText Transport
6 Protocol (HTTP) 1.1, HTTP over Secure Sockets Layer (SSL) (HTTPS), Pipelined
7 HTTP (PHTTP), TCP/IP, FTP, just to name a few.

8 [0048] While an assertion often expresses one particular capability (e.g.,
9 encoding a message in UTF-8), assertions are not limited to expressing only one
10 specific capability. Assertions that specify more than one capability or requirement
11 are referred to as aggregate assertions. An aggregate assertion captures what would
12 otherwise be stated as a set of specific assertions, potentially with preference and
13 choice. For example, a service advertising the WS-I BP 1.0 aggregate assertion
14 could accept messages sent over HTTP or HTTPS, and authenticated with
15 username/password or a client X509 certificate. This is equivalent to a policy
16 containing separate assertions for each of these capabilities.

17 [0049] Groups that exchange messages can define aggregate policy
18 assertions that are shorthand for sets of assertions the groups commonly use. The
19 implementer of the code to process these aggregate assertions may either explicitly
20 expand the shorthand into the more specific assertions, or write code that
21 implements all requirements and capabilities of the aggregate assertion directly. A
22
23
24
25

1 policy can contain either one aggregate assertion or a plurality of assertions, where
2 the assertions can be either specific assertions or aggregate assertions.

3 [0050] It will be appreciated that by using a policy, such as policy 200, a
4 node can specify capabilities, requirements, the number of messages and their
5 form, security measures, reliable messaging, transactions, routing, and other
6 parameters relevant to a message exchange. In addition, policies are extensible,
7 whereby a policy can be extended to include, for example, newly available policy
8 expressions.

9 [0051] Policies are composable, which means that policy expressions having
10 one or more assertions can be inserted into or removed from a policy. Thus, for
11 example, the SOAP header model and Web Services Specifications (WS-specs)
12 outline a composable model, thereby making SOAP headers and WS-specs suitable
13 technologies for implementing a policy scheme outlined herein. In addition, the
14 policy schemes described herein enable nodes to specify a flexible set of protocols
15 at runtime using elements from web services, such as those described by WS-
16 specs.
17
18
19

20 **Exemplary Operations**

21 [0052] Described herein are exemplary methods for implementing dynamic
22 protocol instruction in a network environment. The methods described herein may
23 be embodied as logic instructions on one or more computer-readable medium.
24
25

1 When executed on a processor, the logic instructions cause a general purpose
2 computing device to be programmed as a special-purpose machine that implements
3 the described methods. In the following exemplary operations, the components and
4 connections depicted in the figures may be used to implement dynamic protocol
5 construction in a network environment.

6 [0053] Fig. 3 illustrates a dynamic protocol construction operation flow or
7 algorithm 300 that would be performed by a source node that initiates a message
8 exchange with a destination node. For example, a client accessing a server may
9 execute the operations shown in the operation flow 300. As another example, a
10 first peer attempting to contact a second peer in a peer-to-peer environment may
11 execute the operations shown in Fig. 3 to establish a protocol for communication.
12

13 [0054] The exemplary operations shown and discussed with respect to the
14 dynamic protocol construction operation flow 300 are with respect to a
15 client/server environment, but it is to be understood that the operation flow 300 is
16 generally applicable to any computing device that is initiating a message exchange.
17 In the following description of the operation flow 300, a local policy refers to a
18 policy related to the client and a remote service policy refers to a policy related to a
19 service executing at the server.
20

21 [0055] In a fetching operation 302, the client fetches the remote service
22 policy characterizing capabilities and/or requirements of the server. The fetching
23 operation 302 generally involves retrieving the remote service policy and may
24
25

1 include caching the retrieved service policy at the client for later use. In one
2 implementation of the fetching operation 302, the client sends a request to the
3 server requesting the remote service policy. The client may receive in response an
4 input service policy or an output service policy, or both.

5 [0056] Another implementation of the fetching operation 302 receives the
6 policy or policies incrementally from the server. For example, the client may
7 receive a first set of assertions from the server, followed by a second set, and so
8 on. The client may receive logically related groups of the policy assertions
9 incrementally.
10

11 [0057] In yet another implementation of the fetching operation 302 the
12 client does not fetch the remote service policy from the server related to the remote
13 service policy, but rather the client receives the remote service policy from an
14 advertising server. In this implementation, the remote server advertises the remote
15 server policy on the advertising server, from which the client can access the remote
16 service policy. The client may receive the remote service policy incrementally or
17 all at once.
18

19 [0058] In yet another implementation of the fetching operation 302, the
20 client checks a cache memory on the client for a cached copy of the remote service
21 policy. If the client finds the remote service policy in the client cache, the client
22 may or may not request the remote service policy from the remote server. In this
23 implementation, the client may check the age of the cached remote service policy
24
25

1 (e.g. by comparing a date on the cached remote service policy to a predetermined
2 date) and if the age is greater than a target date, the client may request a new
3 remote service policy from the remote server.

4 [0059] The act of fetching policy may itself be subject to protocol
5 construction. This may involve an initial "bootstrap" protocol, which is agreed to
6 by endpoints based on out of band mechanisms. All implementers that wish to
7 fetch policy will need to agree to use some common protocol or protocols to
8 exchange policy documents. This is the "fetch policy" policy. For example, they
9 may agree to the "fetch policy" policy defined in a paper specification, email each
10 other the "fetch policy" policy for each service, post the "fetch policy" policy on a
11 website for downloading, or advertise the "fetch policy" policy as part of the
12 mechanism used to advertise the service itself. As an example of the later, when a
13 service is advertised in a universal description, discovery, and integration (UDDI)
14 repository, the "fetch policy" policy for the service could be stored with the service
15 address and other capabilities.
16
17

18 [0060] A creating operation 304 creates a local (i.e., client) policy. As
19 discussed above, the local policy can include a number of assertions related to
20 input and output capabilities and requirements of the client. As such, one
21 implementation of the creating operation 304 creates one or more policies based on
22 local hardware and software capabilities, and configuration decisions made by the
23 client implementer and administrator deploying the client. The creating operation
24
25

1 304 preferably creates a local input policy and a local output policy related to the
2 client.

3 [0061] A determining operation 306 determines whether the remote service
4 policy includes at least one set of assertions that are compatible with client
5 capabilities. In one implementation of the determining operation 306, the client
6 identifies one or more groups of assertions in the remote input service policy that
7 intersect with the local (i.e., client) output policy. An intersection between two
8 policies occurs when a group of assertions in the remote input service policy
9 matches or is a subset of a group of assertions in the local output policy.
10

11 [0062] A selecting operation 308 selects one group of assertions from the
12 groups of assertions identified in the determining operation 306, if more than one
13 group of compatible assertions was identified. The selecting operation 308 can
14 consider "preference" values in either the local policy or the remote service policy,
15 or both, to determine if one group of assertions is preferred over another group,
16 and select the preferred group.
17

18 [0063] An implementation of the selecting operation 308 includes
19 configuring software to enable the software to send and receive messages that
20 conform to the selected policy. Configuring the software may involve calling
21 various software modules and/or accessing data stores to obtain security tokens or
22 other data related to the selected policy. The implementation-specific details
23
24
25

1 related to implementing a particular assertion are defined by the specification for
2 the assertion.

3 [0064] An applying operation 310 applies the selected assertions from the
4 selecting operation 308. In one implementation, the applying operation 310 sends
5 a request message to the server. The request message includes an underlying
6 message that the client is sending to the server. For example, if the client is
7 attempting to order books from the server, the request message is a book order
8 message, having service-recognizable syntax and semantics corresponding to a
9 book order.
10

11 [0065] The applying operation 310 appends a header having the selected
12 remote service input policy assertions and the local input policy assertions to the
13 underlying message. By appending the remote service input policy assertions, the
14 client conveys to the remote server the protocols and data formats or other
15 properties that will be used by the client to communicate with the server. By
16 appending the local input policy assertions to the underlying message, the client
17 conveys to the server the properties of the client by which the server can
18 communicate with the client.
19

20 [0066] A receiving operation 312 receives a response from the server. The
21 response from the server is a combination of the underlying response to the client's
22 previous underlying request message and a selected group of assertions from the
23
24
25

1 local input policy. The client is thereby notified about which of the client's
2 communication properties the server will be using to communicate with the client.

3 [0067] An enforcing operation 314 enforces the policy selected by the
4 server. An implementation of the enforcing operation 314 ensures that messages
5 received from the server conform to the input assertions that the server selected
6 previously. Thus, for example, if one of the selected input assertions requires a
7 particular type of encryption, the client will check to see that messages received
8 from the server are encrypted accordingly.

9
10 [0068] Fig. 4 illustrates another dynamic protocol construction operation
11 flow or algorithm 400, which is applicable to a server in a client/server
12 environment. As with the operation flow 300 discussed above, the dynamic
13 protocol construction operation flow 400 of Fig. 4 is not limited to a client/server
14 environment. Rather, the dynamic protocol construction operation flow 400 is
15 applicable to any environment in which a node is the recipient of a request to enter
16 into a message exchange.

17
18 [0069] The dynamic protocol construction operation flow 400 is described
19 from the perspective of the server. As such, the local policy is the policy related to
20 a service executing at the server and the remote policy is the policy related to the
21 client.

22
23 [0070] An advertising operation 402 advertises the local policy of the
24 server. Advertising the local policy involves making the local policy known to at
25

1 least one other node, and in this particular case, the client. One implementation of
2 the advertising operation generates the local policy and transmits all of the local
3 policy to the client in response to a client request for the local policy.

4 [0071] Another implementation of the advertising operation 402 generates
5 the local policy and incrementally transmits the local policy to the client. In this
6 implementation, the server may receive a request from the client for each
7 incremental portion of the local policy, and responsively transmit the requested
8 portion.

9 [0072] Yet another implementation of the advertising operation 402
10 generates the local policy and stores the local policy on an advertising server, from
11 which the client can retrieve the local policy.

12 [0073] In yet another implementation of the advertising operation 402, a
13 copy of the local policy is delivered to a third party service that reads the local
14 policy and the client policy to determine if the two policies are compatible.

15 [0074] A receiving operation 404 receives a message from the client. The
16 message includes an underlying message and a header indicating a group of
17 assertions from the local policy that the client will be using to communicate with
18 the server. The header also provides a remote input policy indicating the client's
19 capabilities and requirements related to receiving messages.

20 [0075] A determining operation 406 determines whether the message
21 received in the receiving operation 404 used a valid policy expression. The
22

1 determining operation 406 checks that the message conforms to the selected group
2 of assertions in the local policy. Thus, for example, the determining operation 406
3 may determine whether the message was encrypted according to an encryption
4 format specified in the local policy. If the message does not conform to the
5 selected group of assertions, then the operation 400 branches "NO" to a notifying
6 operation 408, in which the client is notified that the message does not conform to
7 a valid policy expression.

8 [0076] If, on the other hand, the determining operation 406 does determine
9 that a valid policy expression was applied to the message, the operation 400
10 branches "YES" to a constructing operation 410. The constructing operation 410
11 constructs a receive channel that implements the selected policy expression.
12

13 [0077] A selecting operation 412 selects a policy expression from the
14 remote client's input policy. As indicated above, the selection of a policy
15 expression can be based on the service's capabilities or preferences specified in the
16 input policy and other factors. After the selecting operation 412 selects a policy
17 expression, a generating operation 414 generates a reply message to the client
18 based on the selected policy expression. The reply message typically includes an
19 underlying response message and an indication of the client input policy
20 expression that the service selected for communication with the client.
21
22
23
24
25

Exemplary Computing Device

1 [0078] Fig. 5 is a schematic illustration of an exemplary computing device
2
3 500 that can be utilized to implement a host. Computing device 500 includes one
4 or more processors or processing units 532, a system memory 534, and a bus 536
5 that couples various system components including the system memory 534 to
6 processors 532. The bus 536 represents one or more of any of several types of bus
7 structures, including a memory bus or memory controller, a peripheral bus, an
8 accelerated graphics port, and a processor or local bus using any of a variety of bus
9 architectures. The system memory 534 includes read only memory (ROM) 538 and
10 random access memory (RAM) 540. A basic input/output system (BIOS) 542,
11 containing the basic routines that help to transfer information between elements
12 within computing device 500, such as during start-up, is stored in ROM 538.
13

14 [0079] Computing device 500 further includes a hard disk drive 544 for
15 reading from and writing to a hard disk (not shown), and may include a magnetic
16 disk drive 546 for reading from and writing to a removable magnetic disk 548, and
17 an optical disk drive 550 for reading from or writing to a removable optical disk
18 552 such as a CD ROM or other optical media. The hard disk drive 544, magnetic
19 disk drive 546, and optical disk drive 550 are connected to the bus 536 by
20 appropriate interfaces 554a, 554b, and 554c. The drives and their associated
21 computer-readable media provide nonvolatile storage of computer-readable
22 instructions, data structures, program modules and other data for computing device
23
24
25

1 500. Although the exemplary environment described herein employs a hard disk, a
2 removable magnetic disk 548 and a removable optical disk 552, other types of
3 computer-readable media such as magnetic cassettes, flash memory cards, digital
4 video disks, random access memories (RAMs), read only memories (ROMs), and
5 the like, may also be used in the exemplary operating environment.

6 [0080] A number of program modules may be stored on the hard disk 544,
7 magnetic disk 548, optical disk 552, ROM 538, or RAM 540, including an
8 operating system 558, one or more application programs 560, other program
9 modules 562, and program data 564. A user may enter commands and information
10 into computing device 500 through input devices such as a keyboard 566 and a
11 pointing device 568. Other input devices (not shown) may include a microphone,
12 joystick, game pad, satellite dish, scanner, or the like. These and other input
13 devices are connected to the processing unit 532 through an interface 556 that is
14 coupled to the bus 536. A monitor 572 or other type of display device is also
15 connected to the bus 536 via an interface, such as a video adapter 574.
16
17

18 [0081] Generally, the data processors of computing device 500 are
19 programmed by means of instructions stored at different times in the various
20 computer-readable storage media of the computer. Programs and operating systems
21 may be distributed, for example, on floppy disks, CD-ROMs, or electronically, and
22 are installed or loaded into the secondary memory of the computing device 500. At
23
24
25

1 execution, the programs are loaded at least partially into the computing device's
2 500 primary electronic memory.

3 [0082] Computing device 500 may operate in a networked environment
4 using logical connections to one or more remote computers, such as a remote
5 computer 576. The remote computer 576 may be a personal computer, a server, a
6 router, a network PC, a peer device or other common network node, and typically
7 includes many or all of the elements described above relative to computing device
8 500. The logical connections depicted in Fig. 5 include a LAN 580 and a WAN
9 582. The logical connections may be wired, wireless, or any combination thereof.
10

11 [0083] The WAN 582 can include a number of networks and subnetworks
12 through which data can be routed from the computing device 500 and the remote
13 computer 576, and vice versa. The WAN 582 can include any number of nodes
14 (e.g., DNS servers, routers, etc.) by which messages are directed to the proper
15 destination node.
16

17 [0084] When used in a LAN networking environment, computing device
18 500 is connected to the local network 580 through a network interface or adapter
19 584. When used in a WAN networking environment, computing device 500
20 typically includes a modem 586 or other means for establishing communications
21 over the wide area network 582, such as the Internet. The modem 586, which may
22 be internal or external, is connected to the bus 536 via a serial port interface 556.
23
24
25

1 [0085] In a networked environment, program modules depicted relative to
2 the computing device 500, or portions thereof, may be stored in the remote
3 memory storage device. It will be appreciated that the network connections shown
4 are exemplary and other means of establishing a communications link between the
5 computers may be used.

6 [0086] The computing device 500 may be implemented as a server computer
7 that is dedicated to server applications or that also runs other applications.
8 Alternatively, the computing device 500 may be embodied in, by way of
9 illustration, a stand-alone personal desktop or laptop computer (PCs), workstation,
10 personal digital assistant (PDA), or electronic appliance, to name only a few.

12 [0087] Various modules and techniques may be described herein in the
13 general context of computer-executable instructions, such as program modules,
14 executed by one or more computers or other devices. Generally, program modules
15 include routines, programs, objects, components, data structures, etc. that perform
16 particular tasks or implement particular abstract data types. Typically, the
17 functionality of the program modules may be combined or distributed as desired in
18 various embodiments.
19

20 [0088] An implementation of these modules and techniques may be stored
21 on or transmitted across some form of computer-readable media. Computer-
22 readable media can be any available media that can be accessed by a computer. By
23
24
25

1 way of example, and not limitation, computer-readable media may comprise
2 “computer storage media” and “communications media.”

3 [0089] “Computer storage media” includes volatile and non-volatile,
4 removable and non-removable media implemented in any method or technology
5 for storage of information such as computer-readable instructions, data structures,
6 program modules, or other data. Computer storage media includes, but is not
7 limited to, RAM, ROM, EEPROM, flash memory or other memory technology,
8 CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic
9 cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices,
10 or any other medium which can be used to store the desired information and which
11 can be accessed by a computer.
12

13 [0090] “Communication media” typically embodies computer-readable
14 instructions, data structures, program modules, or other data in a modulated data
15 signal, such as carrier wave or other transport mechanism. Communication media
16 also includes any information delivery media. The term “modulated data signal”
17 means a signal that has one or more of its characteristics set or changed in such a
18 manner as to encode information in the signal. By way of example, and not
19 limitation, communication media includes wired media such as a wired network or
20 direct-wired connection, and wireless media such as acoustic, RF, infrared, and
21 other wireless media. Combinations of any of the above are also included within
22 the scope of computer-readable media.
23
24
25

[0091] In addition to the specific implementations explicitly set forth herein, other aspects and implementations will be apparent to those skilled in the art from consideration of the specification disclosed herein. It is intended that the specification and illustrated implementations be considered as examples only, with a true scope and spirit of the following claims.